

**UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA**

**CAMPUS JUTIAPA**

**FACULTAD INGENIERIA EN SISTEMAS**

**SECCION "A"**

**PROGRAMACION 1**

**ING. RULDIN EFRAIN AYALA RAMOS**



**ANDREA FERNANDA RONQUILLO GODOY 0905-24-15384**

## Implementación en C#

csharp

CopiarEditar

using System;

class Chofer

{

public string Nombre { get; set; }

public int Edad { get; set; }

private string \_tipoLicencia;

public string TipoLicencia

{

get { return \_tipoLicencia; }

set

{

if (Edad < 18 && value != "A")

{

throw new ArgumentException("Un menor de 18 años solo puede tener licencia tipo A.");

}

else if (Edad >= 18 && (value != "A" && value != "B" && value != "C"))

{

```
        throw new ArgumentException("Licencia no válida para esta edad.");
    }

    _tipoLicencia = value;
}

}
```

```
public Chofer(string nombre, int edad, string tipoLicencia)
{
    Nombre = nombre;
    Edad = edad;
    TipoLicencia = tipoLicencia; // Se valida al asignar
}
```

```
public override string ToString()
{
    return $"Chofer {Nombre}, Edad: {Edad}, Licencia: {TipoLicencia}";
}

}
```

```
class Program
{
    static void Main()
    {
```

```

try
{
    Chofer chofer = new Chofer("Juan", 20, "B");

    Console.WriteLine(chofer);

    // Intento de cambio inválido

    chofer.TipoLicencia = "D"; // Esto generará un error
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

```

### Explicación:

1. **Encapsulamiento de Datos:** Se usa una propiedad privada llamada `_tipoLicencia` que guarda el valor de la licencia. A través de la propiedad `TipoLicencia`, que tiene un setter, controlamos cómo se modifica este valor. De esta manera, no se permite cambiar el tipo de licencia directamente sin realizar la validación correspondiente.
2. **Validación al Cambiar la Licencia:** Dentro del setter de la propiedad `TipoLicencia`, se implementan las reglas que validan si el tipo de licencia es adecuado para la edad del chofer. Por ejemplo:
  - Si la persona tiene menos de 18 años, solo puede tener una licencia de tipo "A".

- Si la persona tiene 18 años o más, la licencia debe ser de tipo "A", "B" o "C". Si no se cumple ninguna de estas condiciones, se lanza una excepción.
3. **Uso de Excepciones:** Cuando se intenta asignar un tipo de licencia que no cumple con los requisitos establecidos, el setter lanza una excepción `ArgumentException` con un mensaje que explica el motivo del error. Esto asegura que el programa no continúe con datos inválidos y permite detectar problemas de manera temprana.
  4. **Facilidad de Mantenimiento:** Usar un setter para validar los cambios permite que el código sea más fácil de modificar en el futuro. Si en el futuro se agregan más tipos de licencia o reglas, se pueden ajustar en un solo lugar, evitando duplicación de lógica en diferentes partes del código.

#### **Justificación:**

- **Encapsulación:** Al mantener la propiedad `_tipoLicencia` privada y controlada por el setter, garantizamos que cualquier intento de modificar el tipo de licencia pase por una validación. Esto encapsula la lógica de validación de manera eficiente.
- **Validación Segura:** Cada vez que intentamos cambiar el tipo de licencia, el setter se encarga de verificar si es válido según la edad del chofer. Esto asegura que solo se puedan establecer valores adecuados.
- **Manejo de Errores:** Al lanzar una excepción cuando se intenta asignar un valor inválido, evitamos que el programa continúe con datos incorrectos, lo cual puede prevenir fallos o comportamientos inesperados más adelante.
- **Mantenimiento y Escalabilidad:** Esta estructura hace que el código sea más claro y fácil de mantener. Si en el futuro cambian las reglas para la asignación de licencias, solo será necesario modificar la lógica en el setter, sin necesidad de actualizar múltiples partes del código.

#### **Conclusión:**

El uso de propiedades y validación mediante un setter es una forma eficiente de asegurarse de que los datos de un objeto sean correctos antes de modificarlos. En este caso, se controla que el

tipo de licencia sea apropiado para la edad del chofer, evitando errores y mejorando la claridad del código.

### LLM Utilizado: ChatGPT 4.0

**Prompt Utilizado:** "¿Cuál es la mejor forma de asegurar que cualquier cambio en la licencia sea validado correctamente en C#? Proporciona una solución práctica y bien estructurada."

### Resumen:

El modelo de lenguaje ChatGPT 4.0 proporcionó una solución práctica en C# para validar correctamente el cambio del tipo de licencia en la clase Chofer utilizando encapsulamiento, validación en el setter y el manejo de excepciones. Esto asegura que los datos sean consistentes y se validen adecuadamente, lo que facilita el mantenimiento del código y la expansión futura.

---

### Observaciones y Mejoras

#### 1. Validación de Entrada en el Constructor :

- Aunque ya estás validando la licencia en el constructor mediante el setter, sería bueno añadir validaciones adicionales para **Nombre** y **Edad**. Por ejemplo:
  - **Nombre** no debería ser nulo o vacío.
  - **Edad** debería ser un valor positivo.

#### 2. Tipos de Licencia como Enumeración :

- En lugar de usar cadenas ("A", "B", "C") para representar los tipos de licencia, podrías usar un **enum**. Esto mejora la legibilidad y evita errores tipográficos.

#### 3. Manejo de Excepciones Específicas :

- En lugar de lanzar siempre una **ArgumentException**, podrías crear excepciones personalizadas para diferentes casos (por ejemplo, **LicenciaInvalidaException**).

#### 4. Mensajes de Error Más Descriptivos :

- Los mensajes de error podrían incluir más detalles sobre el contexto, como el nombre del chofer o la edad actual.

## 5. Pruebas Unitarias :

class Chofer

{

public string Nombre { get; private set; }

public int Edad { get; private set; }

private TipoLicencia \_tipoLicencia;

public TipoLicencia TipoLicencia

{

get { return \_tipoLicencia; }

set

{

if (!EsLicenciaValida(value))

{

throw new ArgumentException(\$"La licencia '{value}' no es válida para {Nombre} con {Edad} años.");

}

\_tipoLicencia = value;

}

}

public Chofer(string

## Descripción General

El objetivo es asegurarnos de que cualquier cambio en el tipo de licencia de un chofer sea validado correctamente según su edad. Para lograr esto, utilizamos principios fundamentales de programación orientada a objetos, como el encapsulamiento, la validación centralizada y el manejo adecuado de errores. Esto garantiza que los datos sean consistentes y evita comportamientos inesperados en el programa.

## Cómo Funciona

### 1. Control Centralizado de Datos :

- En lugar de permitir que cualquier parte del código modifique directamente el tipo de licencia, usamos una propiedad (**TipoLicencia**) que actúa como intermediario. Esta propiedad tiene un "setter" (método para asignar valores) que valida si el nuevo valor cumple con las reglas establecidas antes de aceptarlo.
- Por ejemplo, si un chofer menor de 18 años intenta obtener una licencia que no sea de tipo "A", el setter detectará este error y lo bloqueará.

### 2. Validación Basada en Reglas :

- Las reglas son claras:
  - Menores de 18 años solo pueden tener licencias de tipo "A".
  - Mayores de 18 años pueden tener licencias de tipo "A", "B" o "C".
- Estas reglas se implementan dentro del setter de **TipoLicencia**. Si alguien intenta asignar un valor que no cumple con estas condiciones, el programa lanza un error descriptivo.

### 3. Prevención de Errores Temprana :

- Cuando ocurre un intento de asignación inválida, el programa no permite que el error pase desapercibido. En lugar de eso, se genera una excepción que detiene la ejecución y muestra un mensaje claro. Esto ayuda a identificar problemas rápidamente durante el desarrollo y uso del programa.



#### 4. Facilidad para Futuros Cambios :

- Al centralizar las validaciones en el setter, cualquier cambio en las reglas de licencias (como agregar nuevos tipos o modificar las condiciones) solo requiere ajustes en un solo lugar. Esto hace que el código sea más fácil de mantener y menos propenso a errores.

### **Beneficios Clave**

- Protección de Datos : El encapsulamiento asegura que los datos internos de la clase (como el tipo de licencia) no puedan ser modificados sin pasar por las validaciones necesarias.
- Flexibilidad : Si en el futuro se introducen nuevas reglas o tipos de licencia, el código puede adaptarse fácilmente sin afectar otras partes del sistema.
- Claridad : El uso de mensajes de error descriptivos y una estructura bien organizada hace que el código sea más comprensible tanto para el desarrollador como para otros usuarios.
- Seguridad : Al validar cada cambio, evitamos que el programa continúe con datos incorrectos, lo que podría causar problemas graves en escenarios reales.

### **Ejemplo Práctico**

Imagina que tienes un chofer llamado Juan, de 20 años, con una licencia de tipo "B". Si alguien intenta cambiar su licencia a "D" (que no es válida para su edad), el programa no permitirá esta modificación y mostrará un mensaje como:

La licencia 'D' no es válida para Juan con 20 años.

Esto asegura que los datos siempre sean coherentes y que el programa funcione de manera predecible.

## **Conclusión**

En resumen, al usar un setter con validaciones y manejar errores de manera adecuada, logramos un diseño robusto y confiable. Este enfoque no solo protege los datos del chofer, sino que también facilita futuras expansiones y mantenimiento del código. Es una práctica recomendada en programación porque combina claridad, seguridad y flexibilidad en un solo mecanismo.

## **LLM Utilizado: Qwem.ia**

**Prompt:** Necesito una implementación en C# que garantice que los datos de un objeto sean validados antes de ser modificados. Por ejemplo, quiero modelar una clase Empleado donde el salario no pueda ser negativo y la categoría laboral solo pueda ser 'Junior', 'Senior' o 'Manager'. Asegúrate de incluir validaciones robustas, manejo de errores y una explicación del diseño.